LANGUAGE DOCUMENTATION & CONSERVATION

Vol. 14 (2020), pp. 488–502
http://nflrc.hawaii.edu/ldc
http://hdl.handle.net/10125/24948
Revised Version Received: 10 Jul 2020

# SLEXIL: User-centred software for community language documentation

David Beck
*University of Alberta*

Paul Shannon
*Institute for Systems Biology*

SLEXIL (Software Linking ELAN XML to Illuminated Language) is a web application designed to allow users to create animated HTML files from time-aligned transcriptions made in ELAN. Unlike earlier projects with similar goals, SLEXIL is a zero-installation web app developed strictly on user-centred principles, designed with the goal of transferring as much of the technical expertise needed for the process away from the user and onto the maintainers and developers of the software. While SLEXIL itself is rather modest and built for a very specific purpose, we feel that its design is proof of concept for the next generation of user-centred software applications developed for linguists, community language activists, teachers, and others involved in Indigenous and Minority Language Sustainability.

**1. Introduction**[1] The purpose of this paper is to introduce SLEXIL (Software Linking ELAN XML to Illuminated Language), a web application designed to allow users to create animated HTML files from projects made in ELAN (Brugman & Russel 2004; https://archive.mpi.nl/tla/elan/), a common software package for making time-aligned transcriptions of linguistic data. SLEXIL was initially developed to meet a very specific end – the creation of animated HTML versions of single-speaker, audio-only texts – in a way that served the needs of a particular set of users with no specialized computational skills or means of undergoing specialized training (i.e., authors submitting texts to the *International Journal of American Linguistics*). While tools had been developed previously for the task of converting ELAN projects into web pages (e.g., CuPED [Cox & Berez 2010], IATH ETST [Dobrin & Ross 2017], LingView [Pride et al. 2020]), these placed a certain burden on users, requiring the installation of software and/or a certain level of technical sophistication, neither of which could be expected in this particular context of use. SLEXIL was instead developed strictly on user-centred principles (Norman 1986) with the goal of transferring as much of

[1]The authors would like to offer thanks to a number of people who have provided advice, or served as beta testers, in the development of SLEXIL – Chris Cox, Donna Gerdts, Joshua Holden, Jorge Rosés Labrada, Michael Ward, Helen Zhang, and the students in the summer 2019 iterations of LING 423 and 834 of the First Nations Languages Program run by Simon Fraser University. Errors, bugs, and software inadequacies are the responsibility of the authors.



E-ISSN 1934-5275

the technical expertise needed for the process away from the user and onto the maintainers and developers of the software itself. The app was thus designed to meet the following criteria:

1. The software should require no installation by users and, like all current industry-standard web apps, should run in all common web browsers.
2. The software product, in this case an interactive web page, should be ready for use "out-of-the-box" – specifically, it should be ready to view either on a public web server or privately from a user's internet-isolated computer.
3. The software should be built using open source, mainstream software and current web development strategies.
4. The software should be developed in collaborative, open source, agile style, using domain-inspired class design, and be hosted on a public repo such as GitHub, providing unit tests for all classes.

While SLEXIL itself was built for a very specific purpose, we feel that it meets the needs of a wider constituency of users than originally intended, and that its design is proof of concept for the next generation of user-centred software applications developed for linguists, community language activists, teachers, and others involved in Indigenous and Minority Language Sustainability.

**2. Software design** SLEXIL is a web app written in Python using Flask and the plotly/dash libraries (https://plot.ly/dash/), which allow Python code for data visualization and processing to be run through a web browser. Flask itself is a web server written in Python that uses Python HTTP libraries, listens on a port designated by the app for HTTP requests, and responds by returning HTML and JavaScript to a browser. Plotly/dash extends the capabilities of Flask by generating JavaScript and HTML from Python, as well as generating the HTTP protocol commands for exchanging data between a Python web app and the user's browser.

Viewed from the highest level, described in terms of data structures and operations (Knuth 2011), SLEXIL is designed to transform ELAN XML and audio into animated HTML by means of a zero-installation web app, divided into two parts: *server-side code*, which transforms ELAN and audio into an interactive web page for subsequent viewing, and *browser-side code*, which uploads the user's ELAN and audio files, and which allows the user to specify a few processing options. These work together to take a specified *input* – ELAN XML tiered annotation, with audio – and produce a third software entity, the *product* – an interactive, standalone web page version of the original ELAN + audio input. To understand the design of SLEXIL, it is important to understand the distinctions between these entities, and the relationships between them.

The **server-side code** consists of about a dozen Python classes used to represent the entities described in the ELAN file. SLEXIL parses the ELAN document, mapping from that document's tier-structure into these classes, and then translates each class

into an appropriate HTML entity which, when assembled, presents a navigable and interactive HTML web page, with audio, for the entire original text. The server-side code is accompanied by extensive unit tests and a representative variety of ELAN documents for use with these tests.[2] SLEXIL follows the software design principles of "loose coupling" and "separation of concerns": the server-side code runs at the command line, needs no web server, and is easily used and extended by computational linguists. Each Python class has a "toHTML" method. The overarching Text object contains as many Line objects as there are lines in the ELAN file, each Line containing as many "tiers" as the original transcription contains. To create a full interactive web page, "toHTML" is called on the Text object, which then automatically calls "toHTML" recursively on each of the Line and tier objects. If transformations other than HTML are desired (e.g., transformation into XML or another mark-up language), or if a different, possibly more nuanced, HTML is needed, these changes can be factored cleanly into the current design. The current set of toHTML methods makes ample use of recent innovations in HTML and CSS, and creates tabular HTML using the HTML5 <grid> element styled with CSS. This and other HTML tags are conveniently produced using the Python yattag library (https://www.yattag.org). The ELAN file is parsed with the standard Python xml library.

The **browser-side code** consists of HTML, Javascript, and Python/Javascript server/browser communication code. It presents the web page which users see and interact with in order to:

1. upload the ELAN and audio files they wish to transform;
2. set a few options to control that transformation;
3. and download the product, a standalone HTML version of their story.

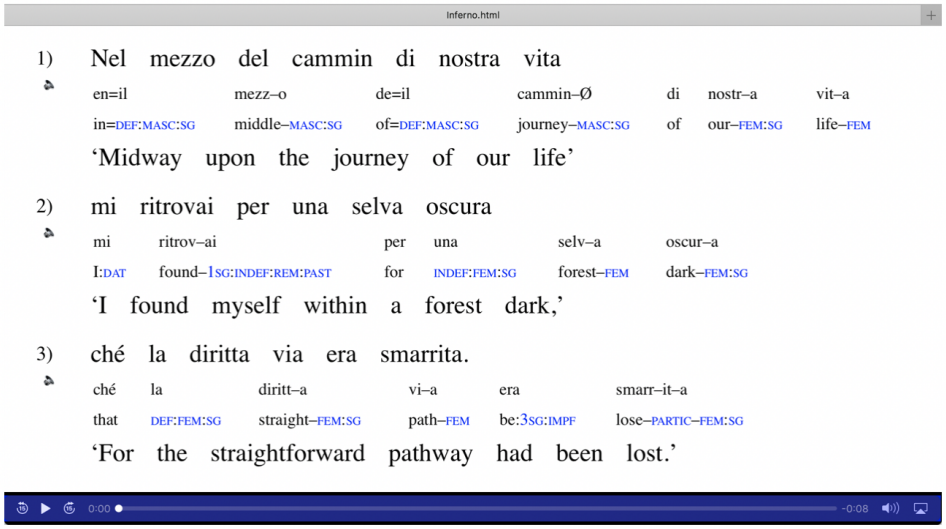
The Python plotly/dash library simplifies the creation of this assemblage, in which HTML and Javascript events in the user's browser must be transmitted to the Python server-side running on the server. The SLEXIL dash web app specifies then creates the browser-side code, and handles all communication between browser and server. Dash was a beta library when SLEXIL began and has recently matured into a reliable strategy for creating interactive web applications written in a single programming language (in this case, Python) with little to no need to learn the intricacies and management of internet inter-process communication protocols or Javascript event handling. This approach has simplified SLEXIL, and allows for future extensions with relative ease.

The third fundamental element of SLEXIL, and its reason for being, is its **product**, a set of HTML and audio files which are a transformation of the ELAN input. This product is a standalone, or web-server offered, interactive web-page rendering of the

[2]A unit test is an automated routine, in this case a Python script, designed to check the behaviour of subunits of an application. These both ensure that particular aspects of the application function as desired during development and, when run as a suite, verify that changes made to the program during the development of one subunit don't adversely affect the behaviour of other subunits. The SLEXIL unit test suite can be run on a local installation from the slexil/tests directory by invoking the included makefile (i.e., by typing "make" on the command line).

original tiered ELAN XML file. This web page presents the text in an interlinearized format with links allowing for the playback of individual lines and an embedded audio player for continuous playback of the recording. During continuous playback, the browser will highlight the transcribed line being spoken and will scroll the screen to follow the playback as it continues.[3] An example of a web page generated with SLEXIL is shown in Figure 1. These pages can be offered over the web for public consumption, or under a password protection scheme. If privacy is required, and/or there is a lack of internet connectivity, the story web page can be viewed as a local file in the user's (unconnected) web browser.

**Figure 1.** Output from SLEXIL using default CSS




Because SLEXIL meditates in this way between data formatted for ELAN and data formatted in HTML, it falls into the category of "middleware" proposed by Kaufman & Finkel (2018). As such, it shares many of the features of software packages such as LingView (Pride et al. 2020) and the web-based apps mentioned above; however, we feel that the user-centred design of SLEXIL has certain advantages that might make this architecture more appealing for a wider range of users, particularly in the context of community-based language programs. Probably the most obvious advantage is that, unlike earlier downloadable (e.g., CuPED [Cox & Berez 2010], LingView [Pride et al. 2020]) and web-based (e.g., Kratylos [Kaufman & Finkel 2018], IATH ETST [Dobrin & Ross 2017], Kwaras/Namuti [Caballero et al. 2019]) applications designed for similar purposes, SLEXIL is a true web app and does not require end users to install any software on their own computer. The basic operation of the program requires only the ability to use a standard web browser and so assumes only the

[3]Examples of files prepared using a precursor of SLEXIL can be found on the *International Journal of American Linguistics* website (https://www.americanlinguistics.org/) under "IJAL online" and "Texts in the Indigenous Languages of the Americas". The specific HTML markup is different but the behaviour of the page is the same.

basic computing skills needed for standard webforms and apps encountered online in people's everyday lives.

An additional advantage of SLEXIL over some other web-based apps like Kratylos (Kafuman & Finkel 2018) and Webonary (SIL International 2017) is that – depending on how the host server is configured – materials uploaded for processing need not be retained for any length of time. The current instance of SLEXIL running at the University of Alberta (https://slexil.artsrn.ualberta.ca), for example, automatically clears the working directory on a regular basis so that uploaded materials do not persist, ensuring that issues of copyright and ownership do not arise. SLEXIL also accepts a reasonably large range of ELAN project file architectures and does not require the user to configure the ELAN project to detailed specifications, as does IATH ETST (Dobrin & Ross 2017). By following the CuPED model (Cox & Berez 2010) and requiring users to provide mappings between ELAN tiers and specific lines in the HTML output, we have made SLEXIL as agnostic as possible with respect to the ELAN project. Finally, the choice of Python and other common software packages supported by a large community of users will help to minimize, to the extent possible, the obsolescence probability of the app itself, thereby avoiding the fate of CuPED, which was in many ways the inspiration for the initial stages of the SLEXIL project. While CuPed was written in Python and took deliberate efforts to use popular open source components, it has been hampered principally by its reliance on FLASH, which has since been superseded by improvements in the open-standard HTML5. By using only mature HTML5, avoiding client-side installation, and separating the computational core from the user interface, SLEXIL software is simpler and might be expected to avoid similar obsolescence traps in the near-to-medium term.

**3. Use-case scenarios** SLEXIL is designed to "backload" as much need for technical expertise as possible away from the end-user and onto the server side of the process, while at the same time remaining open source and customizable by making use of Python for development. We see this architecture as enabling three typical use-case scenarios, each conceived with a particular user-level in mind, running from the novice user with only the most basic computer skills, through a more sophisticated user familiar with command-line applications, to a high-level user with the skills needed to run a web server and deploy an instance of SLEXIL (or something derived from SLEXIL) for use by a particular online constituency. The first group of users we would anticipate to include members of speech communities, language activists, and perhaps even Indigenous Elders, as well as linguists and teachers without technical training, although today many linguists (particularly recent graduate students) have technical skills that would place them in the second group. This second group would also include people with more advanced training in coding and software development who might be able to undertake further development and customization of the app. The third group of users is likely to consist mainly of IT professionals employed by universities and, perhaps, by the better-resourced language programs. Smaller organizations and community groups might not have direct access to third-level users, but could potentially seek this kind of expertise from a larger body such as a university,

particularly in the context of a collaborative project involving a university-affiliated linguist.

**3.1 Running online (user level 1)**[4] Running SLEXIL online requires the user to access a server running an instance of the web app via the address bar in a web browser. The user is then presented with the SLEXIL homepage (Figure 2) and goes through a three-step process to create a project.

**Figure 2.** SLEXIL homepage

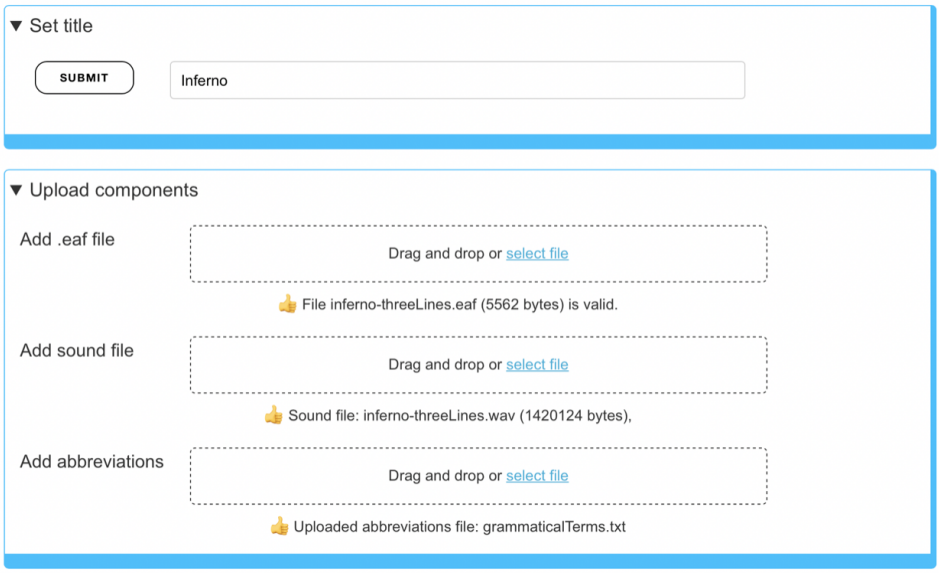

1. The user is asked to enter a title for the project. This is the name that will be given both to the project as a whole and to the HTML file that will display the text.

2. The user opens the "Upload components" tab (Figure 3) and uploads the necessary files. The required elements for a SLEXIL project are, minimally, an ELAN project (.eaf) file and the audio recording on which that file is based. SLEXIL currently only handles projects associated with a single sound file. An optional

[4]A video tutorial on using SLEXIL is available on YouTube at https://youtu.be/7b99pkhQibs. This video, and the screenshots shown in this paper, are based on the demo project provided with SLEXIL, a recording of three lines of Dante's Inferno read by Roberto Benigni. Information about the app and additional details about the components needed to create a project can be found in the "About SLEXIL" tab, a part of any running instance of the app.

third component that can be uploaded at this stage is a plain text list of abbreviations used in interlinear glossing. This list allows SLEXIL to apply a specific CSS style to abbreviations in glosses.

**Figure 3.** Upload components tab



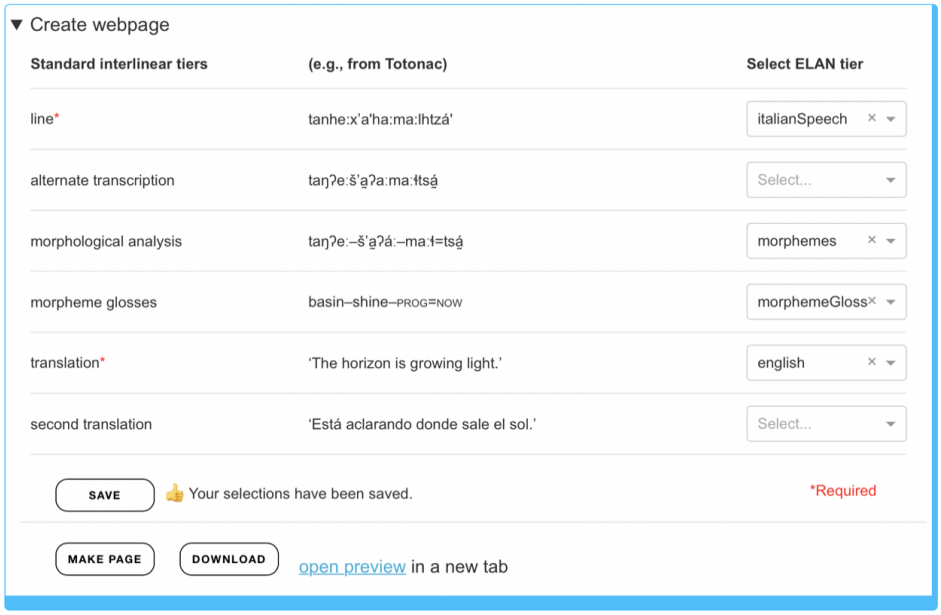

3. The user opens the "Create web page" tab (Figure 4) and uses the pull down menus to tell SLEXIL which tiers in the ELAN project map onto which elements of the HTML model. The user saves these mappings and generates the project by clicking "Make Page". This will generate a preview that can be viewed in the user's browser.[5] If the user is satisfied, the project can be downloaded as a .zip file containing the SLEXIL project.

The third step is the most involved, and the only one that requires any kind of technical knowledge – specifically, a basic understanding of the structure of the ELAN project, or at least the names of the tiers that need to be mapped onto lines of the HTML text. SLEXIL uses the model layout for lines shown in Figure 5 (the example provided is from Upper Necaxa Totonac). Minimally, SLEXIL requires the two lines shown in bold, one for transcribed speech (*line*) and the other a free translation (*translation*).[6] The ELAN project file should be time-aligned to the transcribed speech tier

[5]It should be noted that the security features of some browsers block the playback of embedded media from the page preview, in which case playback will not work as long as the page being accessed is on the remote server. This problem will not persist once the page is downloaded to the user's computer.

[6]It is, however, possible for the annotation on the *translation* tier to be empty. This option was left open to accommodate lines on the speech tier spoken in the contact/working language that don't require glossing. An anonymous reviewer suggests making it possible to process texts that don't have a translation tier, or allowing users not to select that tier for inclusion in the HTML output; this is a good idea and represents a fairly easy fix that we will incorporate into a future iteration of the software.

**Figure 4.** Create web page tab




– that is, it should contain time intervals corresponding to what will be presented as numbered lines in the text. The remaining lines that the user wishes to appear on the web page should be daughters of this tier (either immediate daughters or daughters of another dependent tier). It is not necessary for the speech tier to be the top tier of the ELAN project, and SLEXIL can parse files where the speech-tier annotations are not themselves time-aligned but are instead immediate daughters of time-aligned annotations, as long as these are in a one-to-one relation to the speech tier. It is possible for the project to have tiers that are not to be included in the SLEXIL project, and for these tiers also to be time-aligned. These are simply ignored by SLEXIL if the user does not select them from the pulldown menus.

**Figure 5.** Model for HTML layout

| | | | | | |
|---|---|---|---|---|---|
| **line** | **tanka'lá'kx waká'lh la'hatín misín nakí'wi'** | | | | |
| alternate transcription | tanka̱lá̱kš waká̱ɬ la̱ʔatín misín nakɨ́wɨ̱ | | | | |
| morphological analysis | tanka̱lá̱kš | waká̱ɬ | la̱ʔa–tin | misín | nak=kɨ́wɨ̱ |
| morpheme gloss | head.down | be.high | CLF–one | jaguar | LOC=tree |
| **translation** | **'the jaguar is hanging head-down in a tree'** | | | | |
| second translation | 'el tigre está colgado en un árbol boca abajo' | | | | |

In addition to providing for a simple two-line layout, SLEXIL supports two lines of interlinearization (*morphological analysis* and *morpheme gloss* in Figure 5). The corresponding tiers in the ELAN file can be in a one-to-one relationship to the transcribed speech tier, with words of those tiers separated by tabs, or they can be in a many-to-one relation to the speech tier, with each word and its gloss contained in

their own annotations. Either way, SLEXIL arranges these in a grid layout so that the left edge of each analyzed word is aligned with the left edge of its gloss. If these are not correctly matched (e.g., there are more words on the analysis line than on the gloss line), SLEXIL will alert the user during the next phase of the process as the layout is actually being created. This is a non-fatal error and the user will be able to build the page and then review it for these types of mismatches, which are reported by line number in an error log once the project is downloaded.

SLEXIL also allows for two additional tiers to be displayed, one that will appear immediately below the numbered transcription line (*alternate transcription*) and another that will appear immediately below the free translation (*second translation*). These are in a one-to-one relationship to the numbered line and cannot be subdivided into smaller annotations. The alternate transcription line could be used, as it is in Figure 5, to present a phonemic transcription of a line written in practical orthography, or it could be used to provide a close phonetic transcription of a broad rendering of the text in the line above (or vice versa). The second translation line could be used in the context of a project, like the Totonac project, where the working language of the community (Spanish) is not the language of presentation. It is very likely that users will find other uses for these lines. All six lines in the model are given their own CSS styles, which can be redefined as necessary by editing the .css file included with the project.

The SLEXIL project is downloaded as a .zip file that contains the web page in HTML format, a CSS file specifying the default SLEXIL stylesheet, the JavaScripts needed to run the animation, an errors log, and a directory containing the audio files used during playback. Once unpacked, the .zip file maintains the directory structure needed to view the text and to support line-by-line and animated playback of the text. The user can simply drag the entire folder to wherever they want to store it on their computer and double click on the .html file to open it in a browser for viewing. Users can share files freely across computers and those with the know-how can post the entire project online (maintaining the same directory structure). The CSS is completely customizable, and the SLEXIL project can be adapted with minimum fuss to a content management system such as WordPress.

**3.2 Installing and running on a personal computer (user level 2)** Like IATH ETST (Dobrin & Ross 2017), SLEXIL can be run both online and offline, although offline use requires a slightly higher level of technical skill in that the user must be able to clone the SLEXIL GitHub repo (https://github.com/davidjamesbeck/slexil), and must know how to install the proper version of Python (3.6 or higher) and the following Python packages:

- gunicorn
- xmlschema
- soundfile
- dash 1.01

- dash_table 4.0.1
- dash_core_components 1.0.0
- dash_html_components 1.0.0
- dash-dangerously-set-inner-html 0.0.2
- dash_bootstrap_components
- pandas
- pyyaml
- yattag
- bs4

All of these components are listed in the requirements.txt file in the git repo and can be batch installed using pip ("pip install -r requirements.txt").

Once the local computer is set up, it is necessary to make a small modification to the main program file, webapp4.py. The web app is distributed in a configuration for running on a remote server and needs to be altered slightly for local use. This can be done by scrolling to the end of the file and commenting out the last three lines of the program, then uncommenting the two lines immediately above them. The proper configuration is shown in Figure 6.

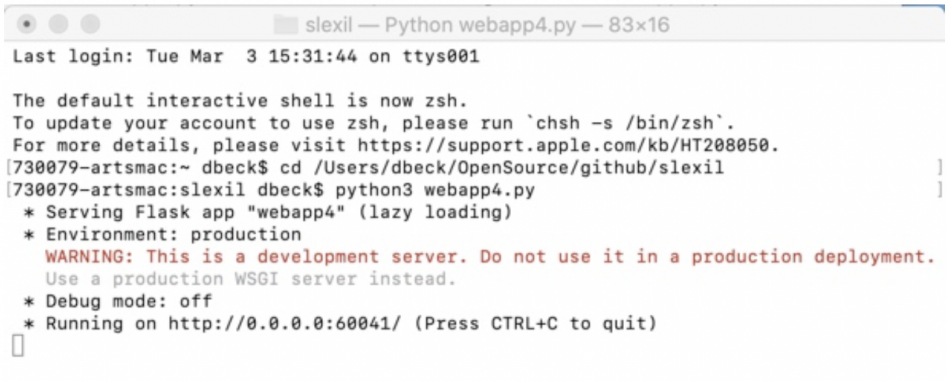
**Figure 6.** Web app configured for running locally

```
932  # ------------------------------------------------------------
933  # enable these lines for running from bash and python
934  if __name__ == "__main__":
935      app.run_server(host='0.0.0.0', port=60041)
936
937  # enable these lines if running with gunicorn
938  # if __name__ == "__main__":
939  #     server = app.server
940  #     app.run()
941
```

Running the app at this point is a matter of opening a terminal window, navigating to the slexil directory, and using Python to run webapp4.py. This will produce the terminal output shown in Figure 7. The next step is to copy the URL given on screen into the address bar of a web browser. This will open an instance of SLEXIL which can be used exactly as described in §3.1 above. The terminal window will display output tracking the user's progress (and any errors, should these occur).

Although the set up for running SLEXIL offline is rather involved, there are some potential advantages in using it on a local computer. This is obviously the case for

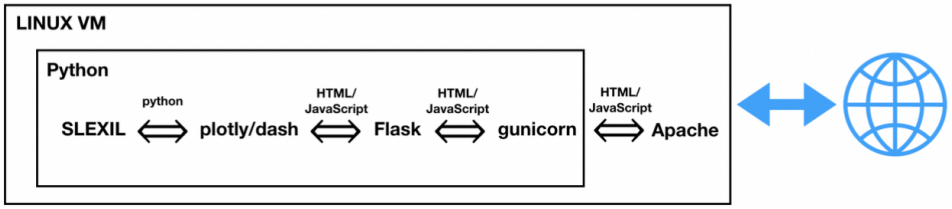
**Figure 7.** Terminal output on launching SLEXIL from the command line




users working in areas where connectivity is poor or in remote communities where it is non-existent, as well as in private homes that don't have reliable wireless networks. People working with private or sensitive materials might also prefer not to use online instances of SLEXIL in order to ensure that materials are not transmitted over the internet or stored on remote servers. Depending on bandwidth, working with large ELAN projects or large recordings online may be slow, in which case running SLEXIL locally may save time, particularly when experimenting with different project configurations or making corrections to optimize presentations. We anticipate, however, that the primary offline users of SLEXIL would be those who want to make some modifications to the app itself, the most obvious (to us) being programmatic changes to the structure of the HTML output. The current HTML model is designed to facilitate the presentation of translated or translated-and-interlinearized texts in the style most favoured by linguists, but this is not necessarily the optimal format for use in the classroom or in the performance of recorded texts for community audiences. Customizing the output format would be well within the reach of someone reasonably familiar with CSS, Python, and object-oriented programming.

**3.3 Deployment of source code (user level 3)** Set up and deployment of SLEXIL to run on a remote web server is a complex process, and the details of the particular deployment will depend on the network being used to support it. The instance at the University of Alberta Arts Research Network runs on a Linux VM in a Python virtual environment, and uses gunicorn and Apache web servers to handle incoming and outgoing HTTP requests from/to external users (Figure 8). Deployment of SLEXIL online requires a level of technical expertise that is likely only to be within reach of organizations like universities, tribal colleges, and other entities that maintain, or have the technical capacity to maintain, their own servers and websites.

**Figure 8.** Server side configuration




Academic and community users may nevertheless be interested in hosting their own instance of SLEXIL in the context of long-term documentation projects that involve community members in the collection and presentation of texts in an easily accessible, non-specialized format. This removes the risk of the particular instance of SLEXIL accessed by the project being taken down by the hosting institution, and provides opportunities to customize the app for specific purposes. With the proper server-side IT support, SLEXIL could, for example, be modified to load projects directly to a world-accessible location on a network, rather than to a user's computer, meaning that completed projects could be turned instantly into a published web page by anyone comfortable enough with computers to follow the procedures outlined in §3.1 above. This would greatly expand the capacity to make recorded materials accessible. There are, no doubt, many other ways in which having a centrally administered application available completely online to a large research or community group would be of benefit. Even though SLEXIL itself may be too narrowly construed for all such projects, the overall architecture of the software seems to present unique advantages and opportunities for research and documentation, as well as collaboration between communities and the institutions that seek to support them.

**4. Perspectives and prospects** Many of the most recent web-based software tools for community language documentation such as Kratylos and Kwaras/Namuti have been developed in the context of projects run by, or in close collaboration with, linguists. These tools are designed to support academic desiderata for gold standard language documentation such as accountability/verifiabilty of data (Himmelmann 2006), standardized and sustainable data structures (Schroeter & Thieberger 2006), and transparency of analysis (Caballero et al. 2019). While these are indisputably laudable goals, and the software packages developed in support of them are by and large excellent, they still present certain challenges from the perspective of the non-linguist and novice end-user. The result has been the creation of a number of first-rate tools for the use of linguists (and those trained by linguists) in support of community language sustainability goals, rather than tools easily used and accessed by community members themselves. Although the usability of software packages is continually improving, we feel that the architecture of SLEXIL represents a logical next step on the path towards genuinely user-centred software for community language documentation.

In its current incarnation, SLEXIL is a fairly specialized tool, but one that could be adapted relatively easily to a broader range of uses, some of which are noted above. Creation of additional HTML models for the presentation of texts, particularly for classroom use, is an obvious extension to SLEXIL functionality, and fairly painless changes could be made to the GUI whereby users would be able to select one or more pre-made output templates. Likewise, given that mastering ELAN is by far the biggest technical hurdle to using SLEXIL, the app could be modified to accept other types of input (say, from FLEx – Black & Simmons 2008 – or TranscriberAG – http://-transag.sourceforge.net), including simple plain text files with time codes presented in a standardized format. Being able to handle recordings of conversations involving multiple speakers would require some modification both on the input side and in the HTML output models, but could nevertheless still be accomplished by extending the existing SLEXIL code.

Projects that involve more than a single audio file would present additional challenges, and would require alteration of both the user interface and the module for audio processing. Accommodating video-recordings would, of course, represent a major advance in functionality and would extend the utility of SLEXIL to sign languages and projects that undertake documentation in video format. In principle, the modular nature of SLEXIL makes this a fairly straightforward programming fix and would involve creating a module to process video (file validation and parsing into clips for individual lines), an alternate HTML model with a video player and a fixed viewing window such as that provided by LingView (Pride et al. 2020), and some modification to the Javascript for playback. More challenging would be issues of file size, processing time, and, for online users, the additional hurdle of bandwidth for data transmission (a problem for third-level users and network administrators to solve). Any of these innovations mentioned here, however, are simply extensions of the current capabilities of SLEXIL and could be implemented by modification of the source code, something well within the capability of developers familiar with Python.

Extending the user-centred model described in this article to more sophisticated tasks, such as those carried out by packages like Kratylos and Kwaras/Namuti, also seems feasible. Indeed, it is likely that the user-centred model makes it possible to produce software that gives us the best of both worlds – apps that are technically sophisticated and operate to the gold standard of language documentation, but at the same time require next to no technical expertise or specialized resources on the part of the end-user, thus maximizing the potential for community participation in language sustainability projects. Recent innovations in software tools for the development of web-based apps such as Flask and plotly/dash provide exciting opportunities for community-based research and the dissemination of both linguistic and computational technology to where it is most needed.

## References

Black, Andrew H. & Gary F. Simons. 2008. The SIL FieldWorks Language Explorer approach to morphological parsing. In Gaylord, Nicholas, Stephen Hilderbrand, Heeyoung Lyu, Alexis Palmer, & Elias Ponvert (eds.), *Texas Linguistics Society 10: Computational linguistics for less-studied languages*, 37–55. CSLI Publications.

Brugman, Hennie & Albert Russel. 2004. Annotating multi-media/multi-modal resources with ELAN. In *Proceedings of the 4th International Conference on Language Resources and Evaluation*. Paris: ELRA/ELDA.

Caballero, Gabriela, Lucien Carrol, & Kevin Mach. 2019. Accessing, managing, and mobilizing an ELAN-based language documentation corpus: The Kwaras and Namuti tools. *Language Documentation & Conservation* 13. 63–82. http://hdl.handle.net/10125/24799.

Cox, Christopher & Andrea Berez. 2010. CuPED – Customizable Presentation of ELAN Documents [Computer software]. https://coxchristopher.github.io/cuped.

Dobrin, Lise M. & Douglas Ross. 2017. The IATH ELAN Text-Sync Tool: A simple system for mobilizing ELAN transcripts on- or off-line. *Language Documentation & Conservation* 11. 94–102. http://hdl.handle.net/10125/24726.

ELAN (Version 5.2). 2018. Nijmegen: Max Planck Institute for Psycholinguistics [Computer software]. https://archive.mpi.nl/tla/elan/. (Accessed 2020-03-03.)

Himmelmann, Nikolaus P. 2006. Language documentation: What is it and what is it good for? In Gippert, Jost, Nikolaus P. Himmelmann, & Ulrike Mosel (eds.), *Essentials of language documentation*, 1–30. Berlin: Walter de Gruyter.

Kaufman, Daniel & Raphael Finkel. 2018. Kratylos: A tool for sharing interlinearized and lexical data in diverse formats. *Language Documentation & Conservation* 12. 124–146. http://hdl.handle.net/10125/24765.

Knuth, Donald E. 2011. *The art of computer programming, vols. 1–4. 3rd edn*. Reading, MA: Addison-Wesley.

Norman, Donald A. 1986. *User-centered system design: New perspectives on human-computer interaction*. Hillsdale: Erlbaum.

Pride, Kalinda, Nicholas Tomlin, & Scott AnderBois. 2020. LingView: A web interface for viewing FLEx and ELAN files. *Language Documentation & Conservation* 14. 87–107. http://hdl.handle.net/10125/24916.

Schroeter, Ronald & Nicholas Thieberger. 2006. EOPAS, the EthnoER online representation of interlinear text. In Barwick, Linda & Nicholas Thieberger (eds.), *Sustainable data from digital fieldwork*, 99–124. Sydney: Sydney University Press.

SIL International. 2017. Webonary [Computer software]. http://www.webonary.org/. (Accessed 2020-03-03).

David Beck
dbeck@ualberta.ca
orcid.org/0000-0001-8422-7728

Paul Shannon
pshannon@systemsbiology.org